SQL (Part 2)

Instructor: David Martin

Reference:
A First Course in Database Systems,
3rd edition, Chapter 6.2 – 6.4

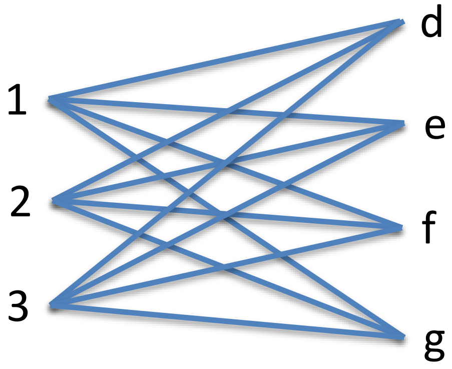
Accessing Canvas

- We use Canvas, not eCommons, for submitting Homeworks and for grading of Homeworks and exams.
  - Login to Canvas at https://canvas.ucsc.edu using your CruzID and Gold password. CMPS 182 should be one of the classes available.
  - Info on Canvas is at http://its.ucsc.edu/canvas/index.html

- Note: we have 3 different systems involved in submissions:
  - Gradiance for online homeworks
  - Canvas for "written" homeworks
  - Unix course lockers for projects
- (Sorry, this was unavoidable)

Review: Cartesian Product

- A: {1,2,3}
- B: {d,e,f,g}
- A × B = { (1,d), (1,e), (1,f), (1,g),
  (2,d), (2,e), (2,f), (2,g),
  (3,d), (3,e), (3,f), (3,g) }



- Suppose that C = {x,y}. What would A x B x C be?

Database Schema for our Examples

- Assume we have the following database schema with five relation schemas.
  - Types are omitted here, but would be required when declaring these tables.

Movies(title, year, length, genre, studioName, producerC#)

MovieStar(name, address, gender, birthdate)

StarsIn(movieTitle, movieYear, starName)

MovieExec(name, address, cert# , netWorth)

Studio(name, address, presC#)

All Pairs of Tuples for These Tables? (Cartesian product, CROSS JOIN)

```
SELECT *
FROM Movies, StarsIn;
```

Movies

Title	Year	Length	Genre	studioName	producerC#
Pretty Woman	1990	119	Romantic	Disney	999
Monster’s Inc.	1990	121	Animation	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

StarsIn

movieTitle	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster’s Inc.	1990	John Goodman

All Pairs of Tuples (Cartesian product, CROSS JOIN)

SELECT *

FROM Movies, StarsIn;

Result

Title	Year	Length	Genre	studioName	Producer C#	movieTitle	Movie Year	starName
Pretty Woman	1990	119	Rom	Disney	999	Pretty Woman	1990	Julia Roberts
Pretty Woman	1990	119	Rom	Disney	999	Monster’s Inc.	1990	John Goodman
Monster’s Inc.	1990	121	Anim	Dreamworks	223	Pretty Woman	1990	Julia Roberts
Monster’s Inc.	1990	121	Anim	Dreamworks	223	Monster’s Inc.	1990	John Goodman
Jurassic Park	1998	145	NULL	Disney	675	Pretty Woman	1990	Julia Roberts
Jurassic Park	1998	145	NULL	Disney	675	Monster’s Inc.	1990	John Goodman

Join With Explicit Equality Condition (1)

```
SELECT *
FROM Movies, StarsIn
WHERE title = movieTitle;
```

Movies

title	year	length	genre	studioName	producerC#
Pretty Woman	1990	119	Romantic	Disney	999
Monster's Inc.	1990	121	Animation	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

StarsIn

movieTitle	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	John Goodman

Join With Explicit Equality Condition (2)

SELECT *

FROM Movies, StarsIn

WHERE title = movieTitle;

Intermediate Result (Temporary Table)

Title	Year	Length	Genre	studioName	Producer C#	movieTitle	Movie Year	starName
Pretty Woman	1990	119	Rom	Disney	999	Pretty Woman	1990	Julia Roberts
Pretty Woman	1990	119	Rom	Disney	999	Monster's Inc.	1990	John Goodman
Monster's Inc.	1990	121	Anim	Dreamworks	223	Pretty Woman	1990	Julia Roberts
Monster's Inc.	1990	121	Anim	Dreamworks	223	Monster's Inc.	1990	John Goodman
Jurassic Park	1998	145	NULL	Disney	675	Pretty Woman	1990	Julia Roberts
Jurassic Park	1998	145	NULL	Disney	675	Monster's Inc.	1990	John Goodman

Join With Equality Condition (3)

```
SELECT *
FROM Movies, StarsIn
WHERE title = movieTitle;
```

Result

Title	Year	Length	Genre	studioName	Producer C#	movieTitle	Movie Year	starName
Pretty Woman	1990	119	Rom	Disney	999	Pretty Woman	1990	Julia Roberts
Monster’s Inc.	1990	121	Anim	Dreamworks	223	Monster’s Inc.	1990	John Goodman

Join With 2-part Condition

SELECT *

FROM Movies, StarsIn

WHERE title = movieTitle AND studioName = ‘Disney’;

Result

Title	Year	Length	Genre	studioName	Producer C#	movieTitle	Movie Year	starName
Pretty Woman	1990	119	Rom	Disney	999	Pretty Woman	1990	Julia Roberts

Disambiguating Attributes

```
SELECT *
FROM Movies, StarsIn
WHERE title = `Pretty Woman’ AND year > 1995;
```

Movies

title	year	length	genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999
Monster’s Inc.	1990	121	true	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

StarsIn

title	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster’s Inc.	1990	John Goodman

What if the first attribute of Movies were called “title”?

Disambiguating Attributes (cont’d)

SELECT *

FROM Movies, StarsIn

WHERE **StarsIn.title** = `Pretty Woman’ AND year > 1995;

Movies

title	year	length	genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999
Monster’s Inc.	1990	121	true	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

StarsIn

title	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster’s Inc.	1990	John Goodman

Tuple Variables:

Another Way to Make Things More Explicit

```
SELECT *
FROM Movies m, StarsIn s
WHERE m.title = s.movietitle;
```

- m and s are *tuple variables*.
- m binds to a tuple (row) in the Movies relation.
- s binds to a tuple (row) in StarsIn relation.

- No different than this:

```
SELECT *
FROM Movies, StarsIn
WHERE title = movietitle;
```

or this:

```
SELECT *
FROM Movies, StarsIn
WHERE Movies.title = StarsIn.movietitle;
```

Self Join

(Using Tuple Variables)

Suppose you want to find pairs of movies released in the same year

```
SELECT *
FROM StarsIn s1, StarsIn s2
WHERE s1.movieYear = s2.movieYear;
```

StarsIn

title	movieYear	starName
Pretty Woman	1990	Julia Roberts
Monster’s Inc.	1990	John Goodman
Star Wars	1977	Harrison Ford
Finding Nemo	2003	Ellen DeGeneres

Self Join

(Using Tuple Variables)

```
SELECT *
FROM StarsIn s1, StarsIn s2
WHERE s1.movieYear = s2.movieYear;
```

Result

s1. title	s1. movieYear	s1. starName	s2. title	s2. movieYear	s2. starName
Pretty Woman	1990	Julia Roberts	Pretty Woman	1990	Julia Roberts
Pretty Woman	1990	Julia Roberts	Monster’s Inc.	1990	John Goodman
Monster’s Inc.	1990	John Goodman	Pretty Woman	1990	Julia Roberts
Monster’s Inc.	1990	John Goodman	Monster’s Inc.	1990	John Goodman
Star Wars	1977	Harrison Ford	Star Wars	1977	Harrison Ford
Finding Nemo	2003	Ellen DeGeneres	Finding Nemo	2003	Ellen DeGeneres

Self Join

(Using Tuple Variables)

```
SELECT *
FROM StarsIn s1, StarsIn s2
WHERE s1.movieYear = s2.movieYear AND s1.title <> s2.title;
```

Result

s1. title	s1. movieYear	s1. starName	s2. title	s2. movieYear	s2. starName
Pretty Woman	1990	Julia Roberts	Monster’s Inc.	1990	John Goodman
Monster’s Inc.	1990	John Goodman	Pretty Woman	1990	Julia Roberts

Self Join

(Using Tuple Variables)

```
SELECT *
FROM StarsIn s1, StarsIn s2
WHERE s1.movieYear = s2.movieYear AND s1.title < s2.title;
```

Result

s1. title	s1. movieYear	s1. starName	s2. title	s2. movieYear	s2. starName
Monster’s Inc.	1990	John Goodman	Pretty Woman	1990	Julia Roberts

Meaning of an SQL Query with Multiple Relations in the FROM Clause

SELECT [DISTINCT] $c_1, c_2, \ldots, c_m$
FROM $R_1, R_2, \ldots, R_n$
[WHERE *condition*]
[ORDER BY <list of attributes>] [DESC]

Suppose we now have more than 1 relation in the FROM clause.

- Let Result begin as an empty set of tuples.
- For every *combination* of tuples t_1 from R_1, t_2 from R_2, …, t_n from R_n
  - if t_1, …, t_n satisfy *condition* (i.e., condition evaluates to true), then add the resulting tuple that consists of c_1, c_2, …, c_m components of *t* into Result.
- If DISTINCT is stated in the SELECT clause, remove duplicates in Result.
- If ORDER BY <list of attributes> exists, order the tuples in Result according to ORDER BY clause.
- Return Result.

SQL Join Expressions

Reference:
A First Course in Database Systems,
3rd edition, Chapter 6.3.6 – 6.3.8

- JOIN
  - ON
  - CROSS JOIN
- NATURAL JOIN
- Outer Joins
  - FULL OUTER JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
- **All of these can appear in the FROM clause**
  - We'll briefly discuss all except OUTER JOIN now
  - All will be discussed further later in the course

JOIN … ON …

Reference:
A First Course in Database Systems,
3rd edition, Chapter 6.3.6 – 6.3.8

R(A,B,C) and S(C,D,E)

- R **JOIN** S **ON** R.B=S.D **AND** R.A=S.E;
  - Selects only tuples from R and S where R.B=S.D and R.A=S.E
  - Schema of the resulting relation:
    (R.A, R.B, R.C, S.C, S.D, S.E);
  - Equivalent to:

```
SELECT *
FROM R, S
WHERE R.B=S.D AND R.A=S.E;
```

CROSS JOIN

Reference:
A First Course in Database Systems,
3rd edition, Chapter 6.3.6 – 6.3.8

R(A,B,C) and S(C,D,E)

- R **CROSS JOIN** S;
  - Product of the two relations R and S.
  - Schema of resulting relation:
    (R.A, R.B, R.C, S.C, S.D, S.E).
  - Equivalent to:

```
SELECT *
FROM R, S;
```

NATURAL JOIN

R(A,B,C) and S(C,D,E)

- R **NATURAL JOIN** S;
  - Schema of the resulting relation: (A, B, C, D, E)
  - Equivalent to:

```
SELECT R.A, R.B, R.C, S.D, S.E
FROM R, S
WHERE R.C = S.C;
```

Review: Sets vs. Bags (or Multisets)

From basic set theory –

- Every element in a set is distinct
  - E.g., {2,4,6} is a set but {2,4,6,2,2} is not a set.

- A bag (or multiset) may contain repeated elements.
  - E.g., {{2,4,6}} is a bag. So is {{2,4,6,2,2}}.
    - Note that double set brackets in {{2,4,6}} indicate it’s a bag, not a set
  - Equivalently written as {{2[3],4[1],6[1]}}.

- The order among elements in a set or bag is not important
  - E.g., {2,4,6} = {4,2,6} = {6,4,2}
  - {{2,4,6,2,2}} = {{2,2,2,6,4}} = {{6,2,2,4,2}}.

Set and Bag Operations in SQL

- Set Union, Set Intersection, Set Difference
- Bag Union, Bag Intersection, Bag Difference

- Other set/bag operations
  - IN, NOT IN, op ANY, op ALL, EXISTS, NOT EXISTS
  - More on these later.

Reference:
A First Course in Database Systems,
3rd edition, Chapter 6.2.5, 6.4.1, 6.4.2

Set Union

R(A,B,C), S(A,B,C)

- Input to union must be *union-compatible*.
  - R and S must have the same set of attributes and the corresponding attributes must be of the same type.
- Output of union has the same schema as R or S.
- Meaning: Output consists of the **set** of all tuples from R and from S.
  - UNION could be called UNION DISTINCT

```
(SELECT *
FROM R)
UNION
(SELECT *
FROM S);
```

```
(SELECT *
FROM R
WHERE A > 10)
UNION
(SELECT *
FROM S
WHERE B < 300);
```

Bag Union

R(A,B,C), S(A,B,C)

- Input to union must be *union-compatible*.
  - R and S must have attributes of the same types in the same order.
- Output of union has the same schema as R or S.
- Meaning: Output consists of the collection of all tuples from R and from S, including duplicate tuples.
  - *Subtlety: Attributes/column names may be different; R’s are used.*

```
(SELECT *
FROM R)
UNION ALL
(SELECT *
FROM S);
```

```
(SELECT *
FROM R
WHERE A > 10)
UNION ALL
(SELECT *
FROM S
WHERE B < 300);
```

Intersection; Difference (INTERSECT; EXCEPT)

- Like union, intersection and difference are binary operators.
  - Input to intersection/difference operator consists of two relations R and S, and they must be union-compatible.
  - Output has the same type as R or S.

- Set Intersection, Bag Intersection
  - <Query1> **INTERSECT** <Query2>, <Query1> **INTERSECT ALL** <Query2>
  - Find all tuples that are in the results of both Query1 and Query2.

- Set Difference, Bag Difference
  - <Query1> **EXCEPT** <Query2>, <Query1> **EXCEPT ALL** <Query2>
  - Find all tuples that are in the result of Query1, but not in the result of Query2.

Operator Precedence

- <Query1> EXCEPT <Query2> EXCEPT <Query3> means
  (<Query1> EXCEPT <Query2>) EXCEPT <Query3>

Order of operations originally was: UNION, INTERSECT and EXCEPT have the same priority, and are executed left-to-right.

But this changed in the SQL standard, and has changed in most implementations!
Now, INTERSECT has a higher priority than UNION and EXCEPT
(just as * has a higher priority than + and -) , so:

Query1 UNION Query2 INTERSECT Query3

would be executed as:

Query1 UNION (Query2 INTERSECT Query3)

not as:

(Query1 UNION Query2) INTERSECT Query3

Subqueries

Reference:
A First Course in Database Systems, 3rd edition, Chapter 6.3.

- A subquery is a query that is embedded in another query.
  - Note that queries with UNION, INTERSECT, and EXCEPT have two subqueries.

- A subquery can return a constant (scalar value) which can be compared against another constant in the WHERE clause, or can be used in a boolean expression.

- A subquery can also return a relation.

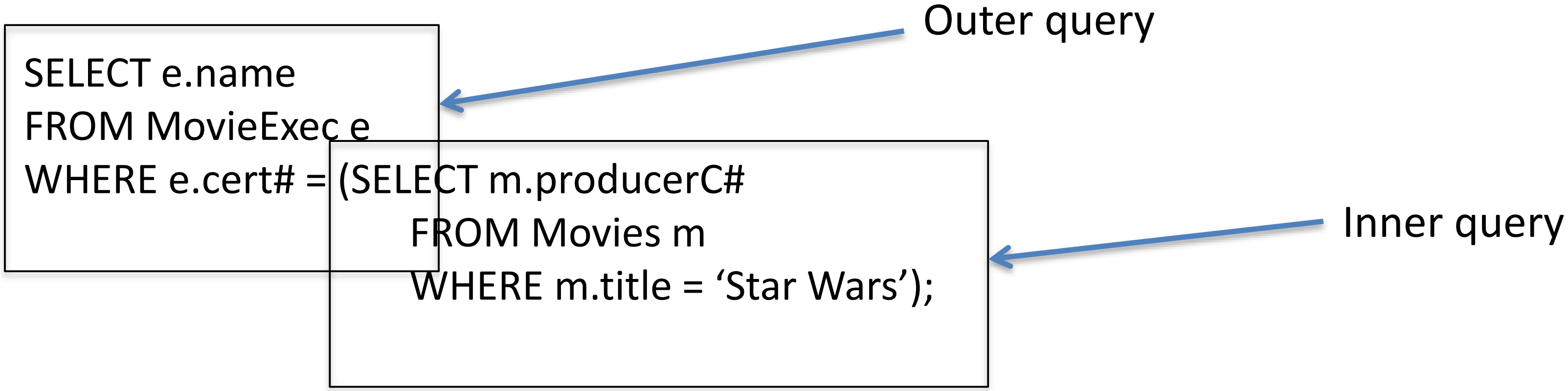
- A subquery returning a relation can appear in the FROM clause, followed by a tuple variable that refers to the tuples in the result of the subquery
  - … just as a tuple variable can be used to refer to a table.

Subqueries that Return Scalar Values

Movies(title, year, length, genre, studioName, producerC#)
MovieExec(name, address, cert#, netWorth), **with name UNIQUE**

- Find names of all executives who produced the movie ‘Star Wars’.
  - Careful: Don’t write query the second way—could get a runtime error!

```
SELECT e.name
FROM Movies m, MovieExec e
WHERE m.title=‘Star Wars’ AND m.producerC# = e.cert#;
```



Outer query

```
SELECT e.name
FROM MovieExec e
WHERE e.cert# = (SELECT m.producerC#
          FROM Movies m
          WHERE m.title = ‘Star Wars’);
```

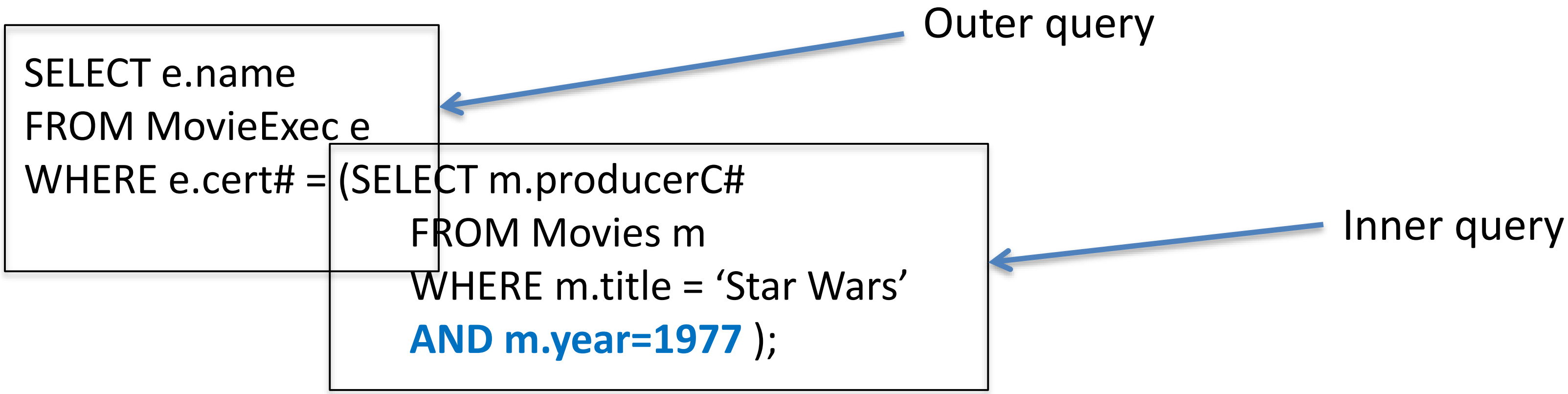
Inner query

Subqueries that Return Scalar Values

Movies(title, year, length, genre, studioName, producerC#)
MovieExec(name, address, cert#, netWorth), **with name UNIQUE**

- Find names of all executives who produced the 1977 movie ‘Star Wars’.
  - **Won’t get a runtime error**, since you expect to get one tuple from subquery

```
SELECT e.name
FROM Movies m, MovieExec e
WHERE m.title=‘Star Wars’ AND m.year=1977 AND m.producerC# = e.cert#;
```



```
SELECT e.name
FROM MovieExec e
WHERE e.cert# = (SELECT m.producerC#
        FROM Movies m
        WHERE m.title = ‘Star Wars’
        AND m.year=1977 );
```

Outer query

Inner query

Subqueries that Return Relations

```
SELECT e.name
FROM MovieExec e
WHERE e.cert#  IN (SELECT m.producerC#
                   FROM Movies m
                   WHERE m.title = ‘Star Wars’);
```

- **IN, NOT IN**

Is this query equivalent to the one above?

```
SELECT e.name
FROM MovieExec e, Movies m
WHERE m.title = ‘Star Wars’
    AND m.producerC# = e.cert#;
```

Having Subquery that Returns a Relation in the FROM Clause

```
SELECT e.name
FROM MovieExec e,
                    (SELECT m.producerC#
                     FROM Movies m
                     WHERE m.title = ‘Star Wars’) p
 WHERE e.cert# = p.producerC# ;
```

Is this query equivalent to the one above?

```
SELECT e.name
FROM MovieExec e, Movies m
WHERE e.cert# = m.producerC#
    AND m.title = ‘Star Wars’;
```

Subqueries with Subqueries

What does this query do? **(Assume that MovieExec.name is UNIQUE.)**

```
SELECT e.name
FROM e.MovieExec
WHERE e.cert#  IN (SELECT m.producerC#
                   FROM m.Movies
                   WHERE (m.title, m.year) IN (SELECT s.movieTitle, s.movieYear
                                               FROM StarsIn s
                                               WHERE s.starName = ‘Harrison Ford’)
                   );
```

Is this query equivalent?

```
SELECT e.name
FROM MovieExec e, Movies m, StarsIn s
WHERE e.cert# = m.producerC# AND m.title = s.movieTitle
  AND m.year = s.movieYear AND s.starName = ‘Harrison Ford’;
```

Correlated Subqueries

- In all the examples so far, the inner query has been <u>independent</u> of the outer query.
- An inner query can also <u>depend on</u> attributes in the outer query; that's called **correlation**.

Find the movie titles that have been used for two or more movies.

```
SELECT DISTINCT m.title
FROM Movies m
WHERE m.year < ANY (SELECT m2.year
                    FROM Movies m2
                    WHERE m2.title = m.title);
```

DISTINCT needed because title may been been used for three or more movies!

Correlation via tuple variable m

Checks that year is less than at least one of the answers returned by the subquery.
< ANY, <= ANY, > ANY, >= ANY, <> ANY, = ANY
< ALL, <= ALL, > ALL, >= ALL, <> ALL, = ALL

Correlated Subqueries (cont’d)

```
SELECT starName
FROM StarsIn s
WHERE EXISTS (SELECT *
              FROM StarsIn c
              WHERE s.movieTitle <> c.movieTitle AND
                    s.movieYear = c.movieYear AND
                    s.starName = c.starName);
```

Checks that the subquery returns a non-empty result.
Can write EXISTS or NOT EXISTS.

Set Comparison Operators

- x IN Q
  - Returns true if x occurs in the collection Q.
- x NOT IN Q
  - Returns true if x does not occur in the collection Q.
- EXISTS Q
  - Returns true if Q is a non-empty collection.
- NOT EXISTS Q
  - Returns true if Q is an empty collection.

Set Comparison Operators (cont'd)

- *x op* ANY Q, x *op* ALL Q in WHERE clause
  - x is a scalar expression
  - Q is a SQL query
  - *op* is one of { <, <=, >, >=, <>, = }.

- x *op* ANY Q
  - What does this mean?
  - SOME can be used instead of ANY

- x op ALL Q
  - What does this mean?

Subqueries in the FROM Clause

- Find the names of all movie executives who produced movies that Harrison Ford acted in.

```
SELECT e.name
FROM MovieExec e, (SELECT m.producerC#
                   FROM Movies m, StarsIn s
                   WHERE m.title = s.movieTitle AND
                         m.year = s.movieYear AND
                         s.starName = ‘Harrison Ford’) p
WHERE e.cert# = p.producerC#;
```

```
Movies(title, year, length, genre, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieExec(name, address, cert#, netWorth)
```

Things To Remember About Subqueries

- Their result is simply a table
  - As indicated by their SELECT clause
  - Special case: a single-cell table is treated as a scalar
- Can appear in FROM and/or WHERE clauses
- To understand them: think from the inside out
- With correlated queries, imagine you are looping over the tuples of the outer query and passing values to the inner query
- Often can be done in a different way
  - With no subqueries but a more complex WHERE clause